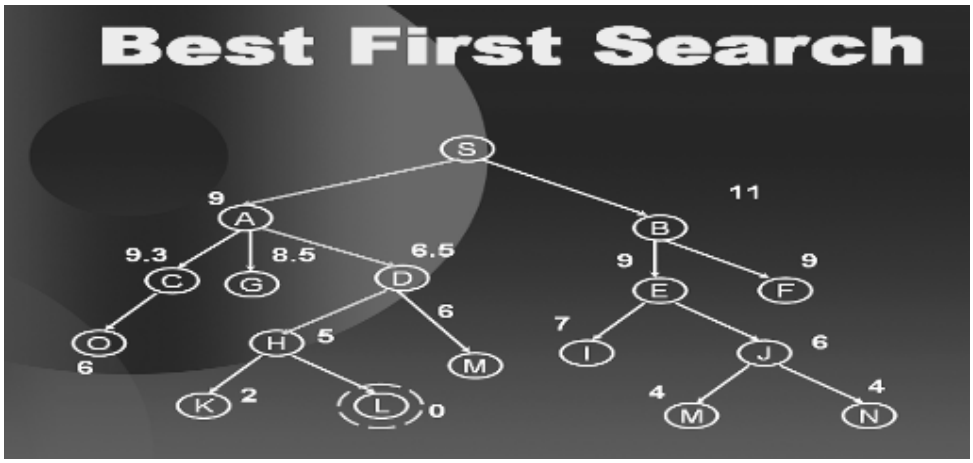


At D we have S, G, B, H, M and J as the options. We select H which is the best of them.

At last from H we find L as the best. Hence best first search is a greedy approach will looks for the best amongst the available options and hence can sometimes reduce the searching time. All these heuristically informed procedures are considered better but they do not guarantee the optimal solution, as they are dependent on the quality of heuristic being used.

2.18 Optimal Searches

So far we have looked at uninformed and informed searches. Both have their advantages and disadvantages. But one thing that lacks in both is that whenever they find a solution they immediately stop. They never consider that their might be more than one solution to the problem and the solution that they have ignored might be the optimal one.

A simplest approach to find the optimal solution is this; find all the possible solutions using either an uninformed search or informed search and once you have searched the whole search space and no other solution exists, then choose

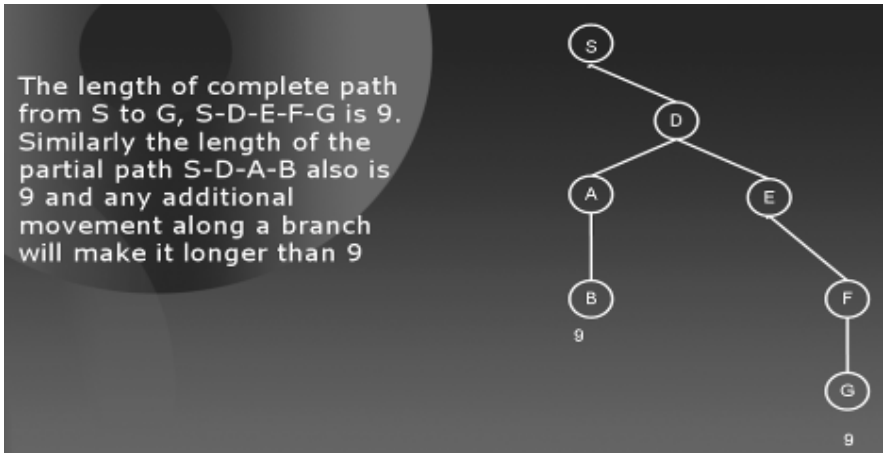
the most optimal amongst the solutions found. This approach is analogous to the brute force method and is also called the British museum procedure.

But in reality, exploring the entire search space is never feasible and at times is not even possible, for instance, if we just consider the tree corresponding to a game of chess (we will learn about game trees later), the effective branching factor is 16 and the effective depth is 100. The number of branches in an exhaustive survey would be on the order of 10^{120}. Hence a huge amount of computation power and time is required in solving the optimal search problems in a brute force manner.

2.19 Branch and Bound

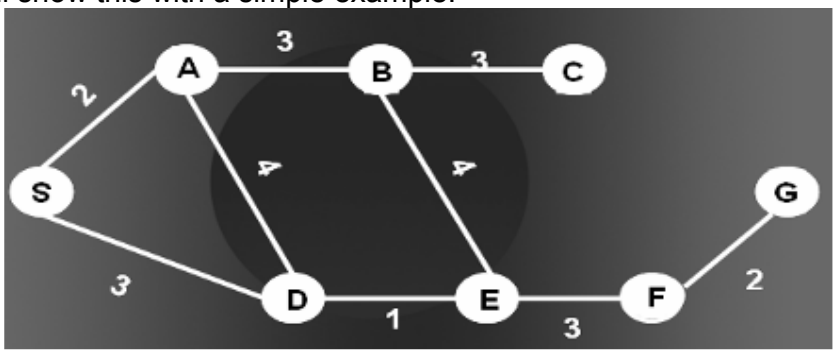
In order to solve our problem of optimal search without using a brute force technique, people have come up with different procedures. One such procedure is called branch-and-bound method.

The simple idea of branch and bound is the following:

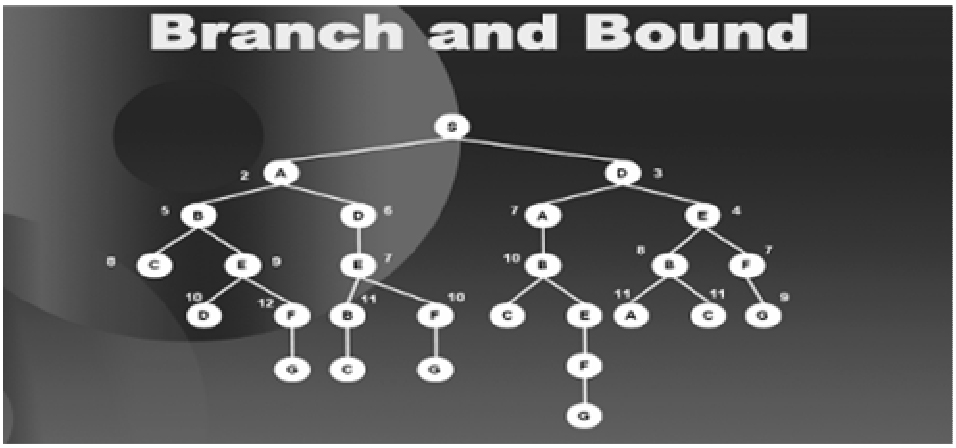


The length of the complete path from S to G is 9. Also note that while traveling from S to B we have already covered a distance of 9 units. So traveling further from S D A B to some other node will make the path longer. So we ignore any further paths ahead of the path S D A B.

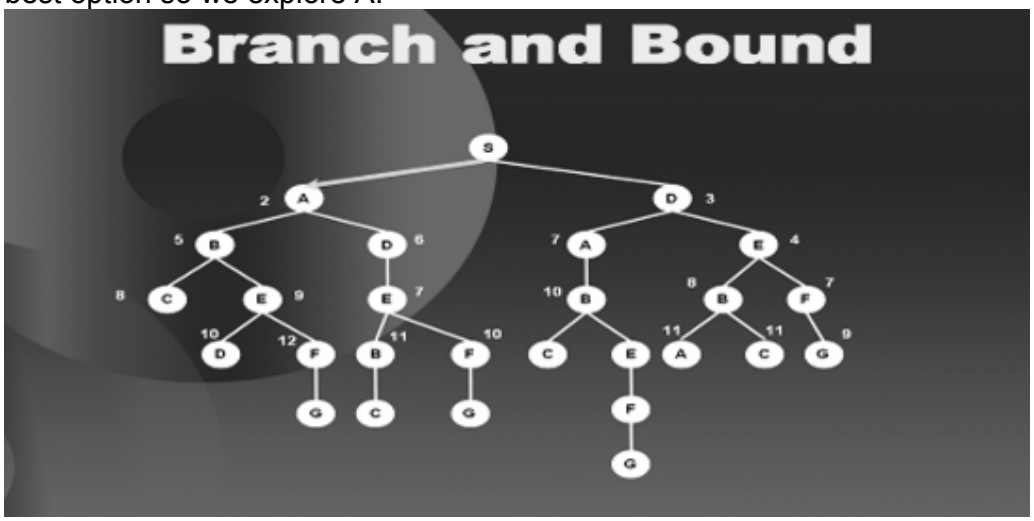
We will show this with a simple example.



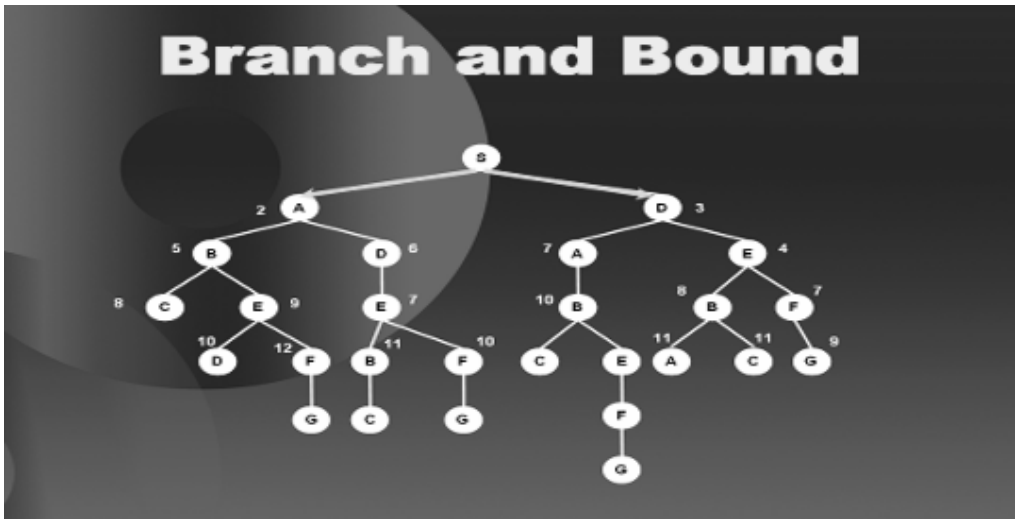
The diagram above shows the same city road map with distance between the cities labels on the edges. We convert the map to a tree as shown below.



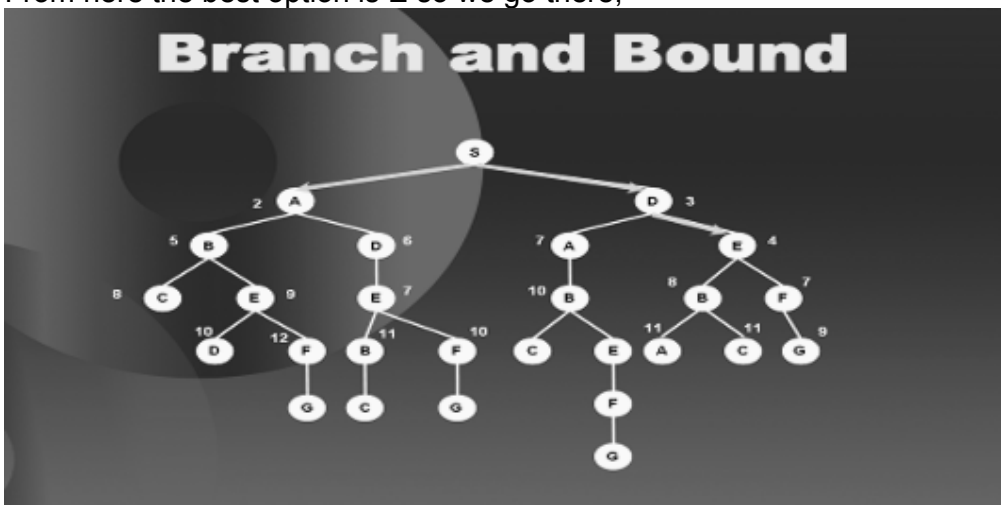
We proceed in a Best First Search manner. Starting at S we see that A is the best option so we explore A.



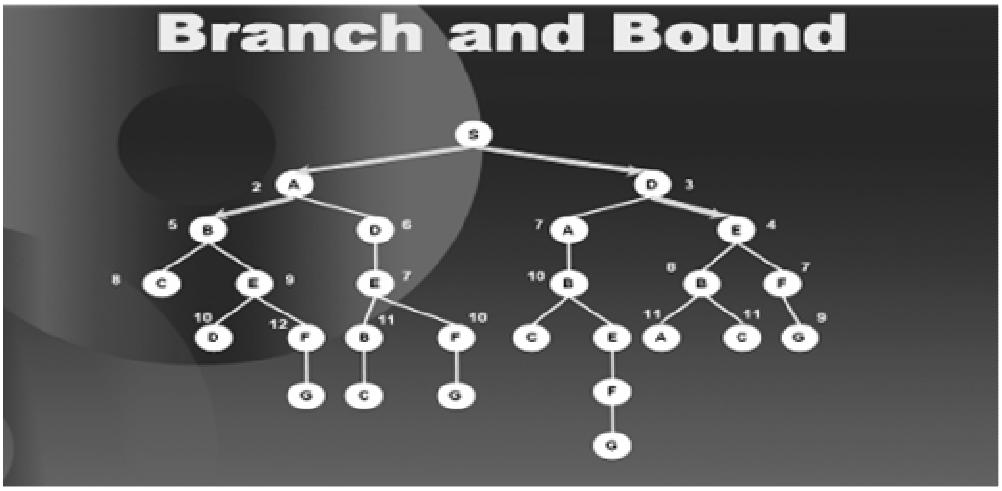
From S the options to travel are B and D, the children of A and D the child of S. Among these, D the child of S is the best option. So we explore D.



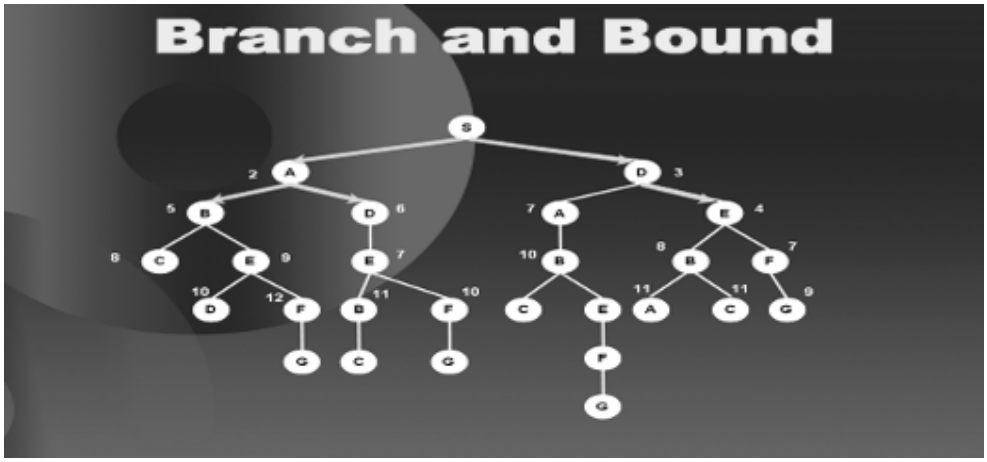
From here the best option is E so we go there,



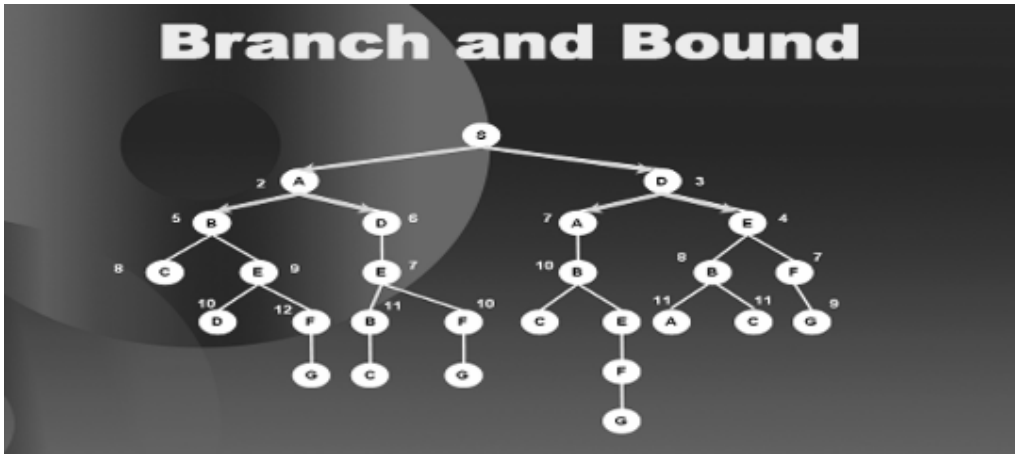
then B,



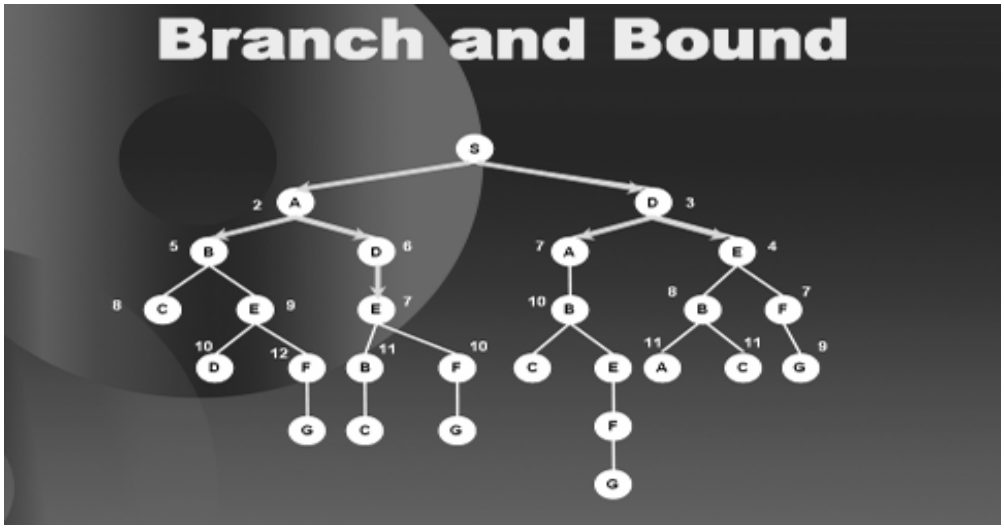
then D,



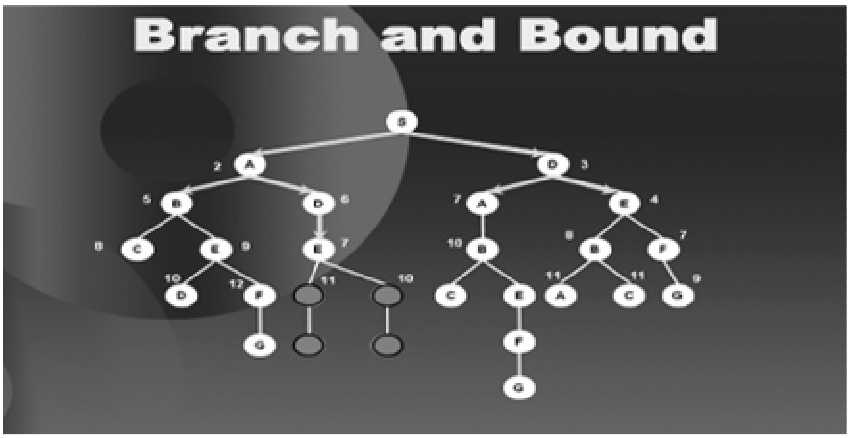
Here we have E, F and A as equally good options so we select arbitrarily and move to say A,



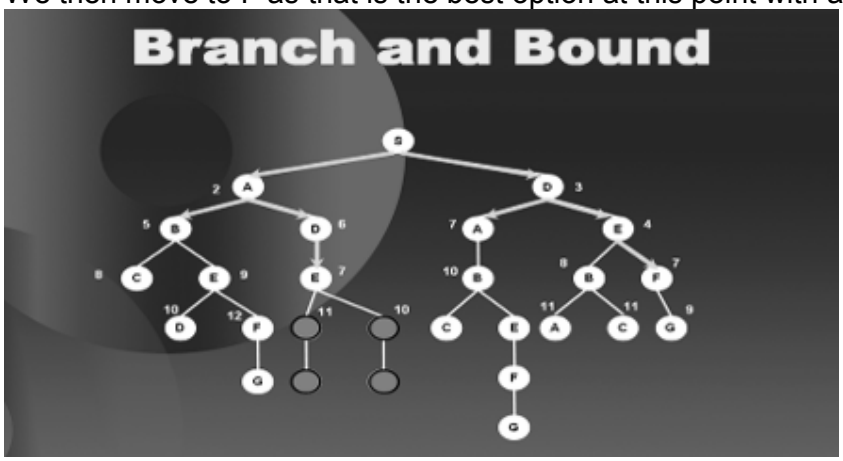
then E.



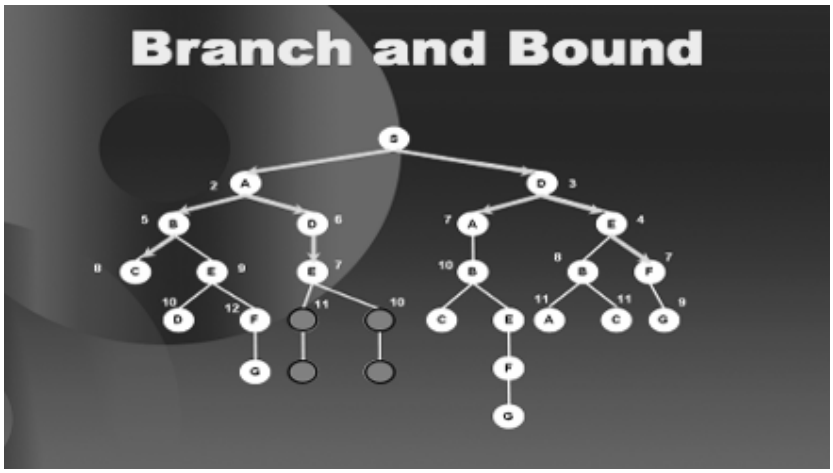
When we explore E we find out that if we follow this path further, our path length will increase beyond 9 which is the distance of S to G. Hence we block all the further sub-trees along this path, as shown in the diagram below.



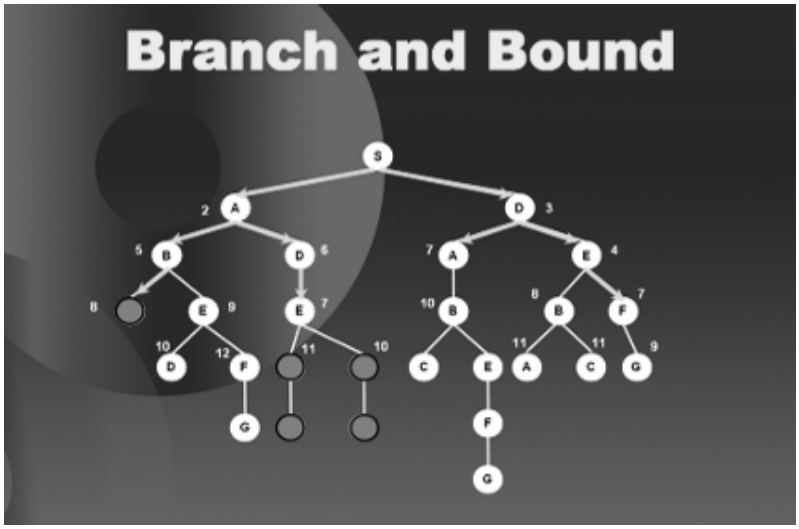
We then move to F as that is the best option at this point with a value 7.



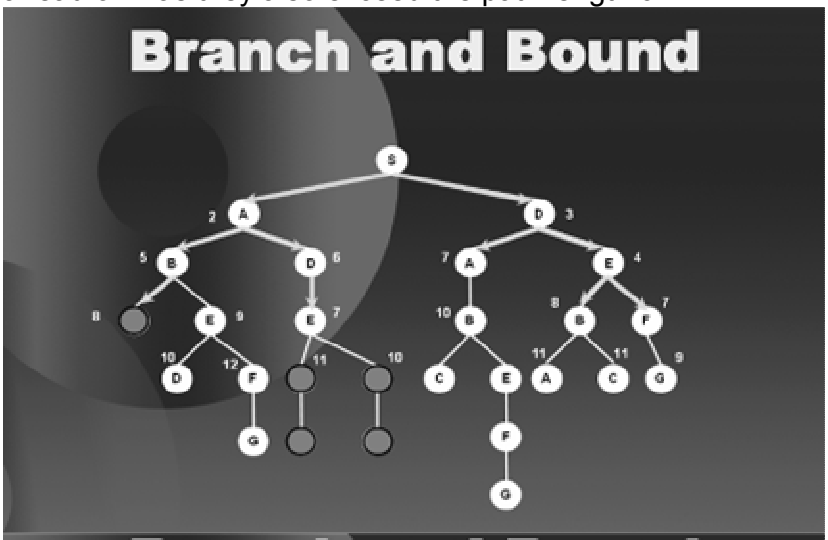
then C,

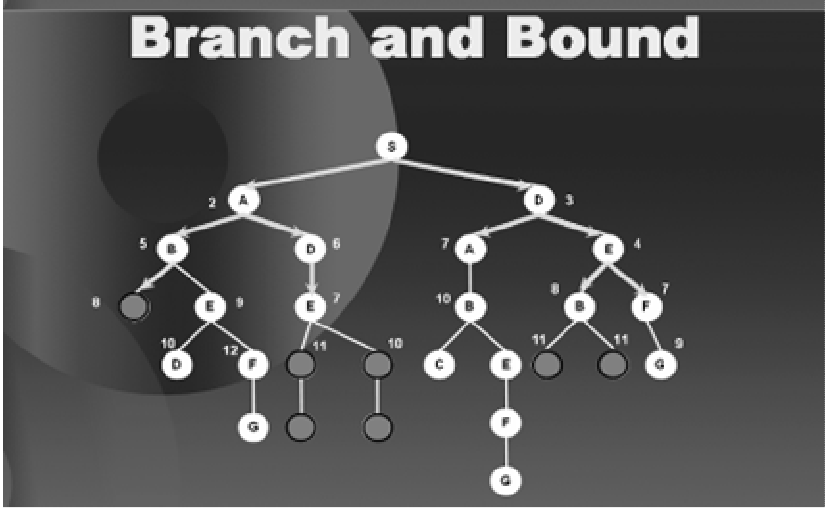


We see that C is a leaf node so we bind C too as shown in the next diagram.

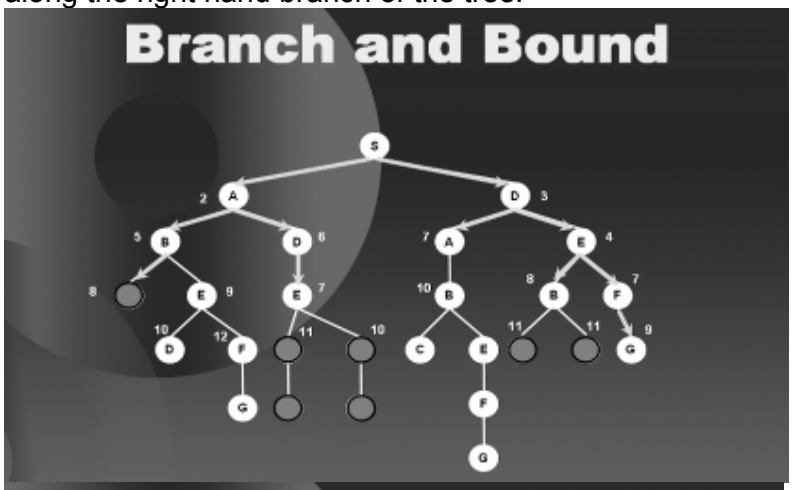


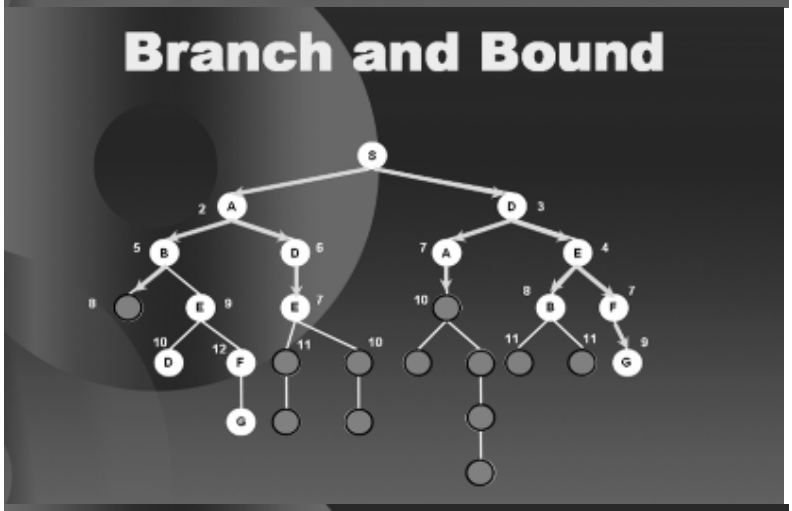
Then we move to B on the right hand side of the tree and bind the sub trees ahead of B as they also exceed the path length 9.

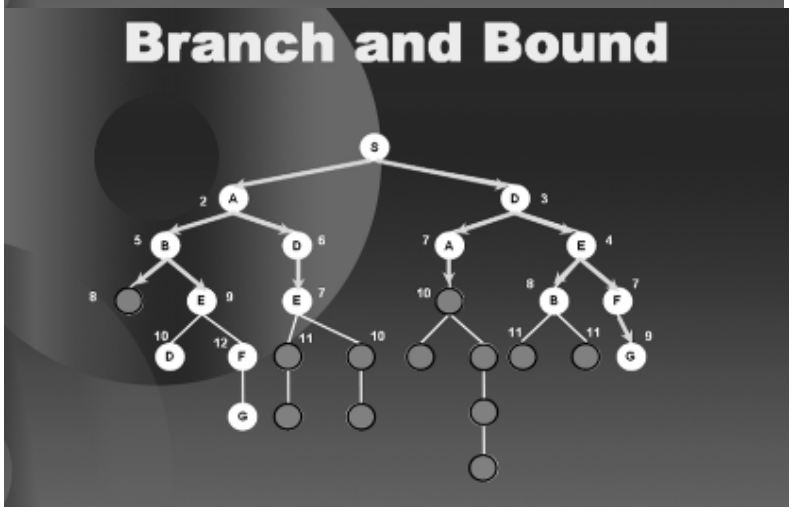


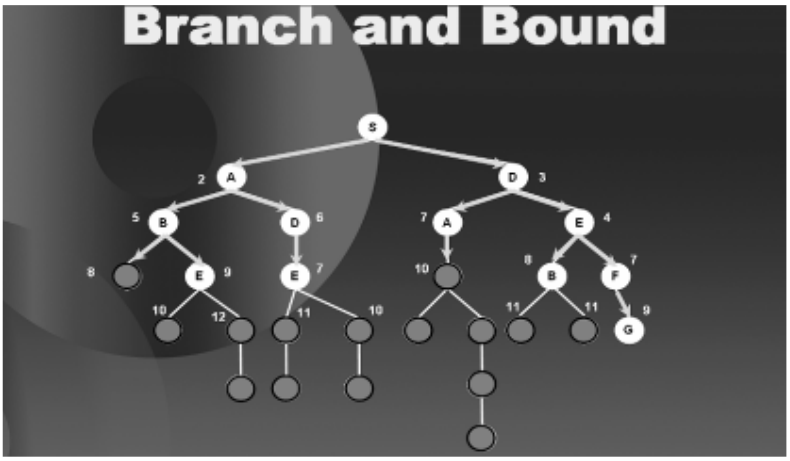


We go on proceeding in this fashion, binding the paths that exceed 9 and hence we are saved from traversing a considerable portion of the tree. The subsequent diagrams complete the search until it has found all the optimal solution, that is along the right hand branch of the tree.









Notice that we have saved ourselves from traversing a considerable portion of the tree and still have found the optimal solution. The basic idea was to reduce the search space by binding the paths that exceed the path length from S to G.

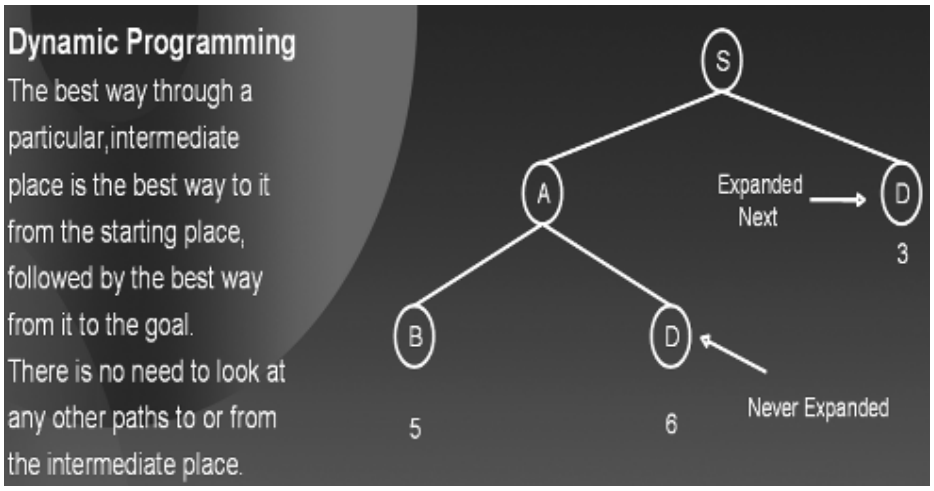
2.20 Improvements in Branch and Bound

The above procedure can be improved in many different ways. We will discuss the two most famous ways to improve it.

1. Estimates
2. Dynamic Programming

The idea of estimates is that we can travel in the solution space using a heuristic estimate. By using “guesses” about remaining distance as well as facts about distance already accumulated we will be able to travel in the solution space more efficiently. Hence we use the estimates of the remaining distance. A problem here is that if we go with an overestimate of the remaining distance then we might loose a solution that is somewhere nearby. Hence we always travel with underestimates of the remaining distance. We will demonstrate this improvement with an example.

The second improvement is dynamic programming. The simple idea behind dynamic programming is that if we can reach a specific node through more than one different path then we shall take the path with the minimum cost.



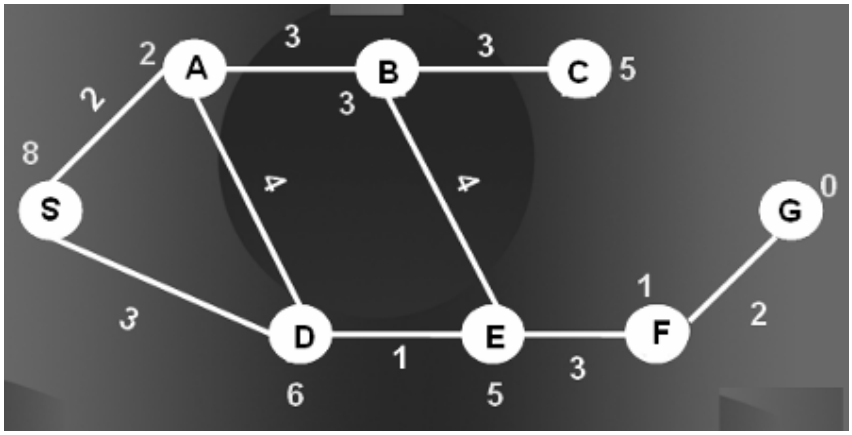
In the diagram you can see that we can reach node D directly from S with a cost of 3 and via S A D with a cost of 6 hence we will never expand the path with the larger cost of reaching the same node.

When we include these two improvements in branch and bound then we name it as a different technique known as A* Procedure.

2.21 A Procedure*

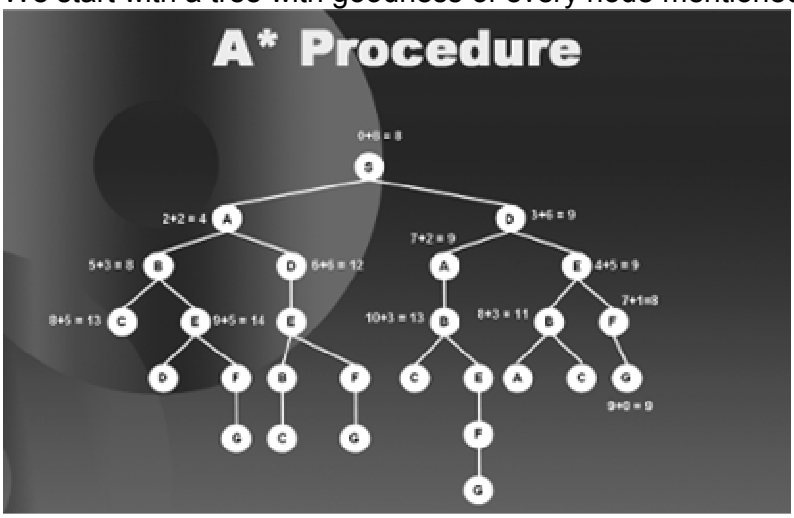
This is actually branch and bound technique with the improvement of underestimates and dynamic programming.

We will discuss the technique with the same example as that in branch-and-bound.

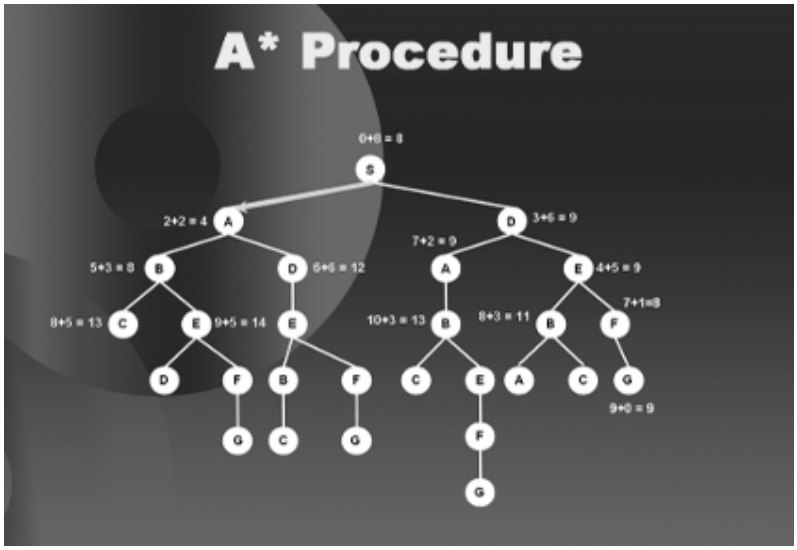


The values on the nodes shown in yellow are the underestimates of the distance of a specific node from G. The values on the edges are the distance between two adjacent cities.

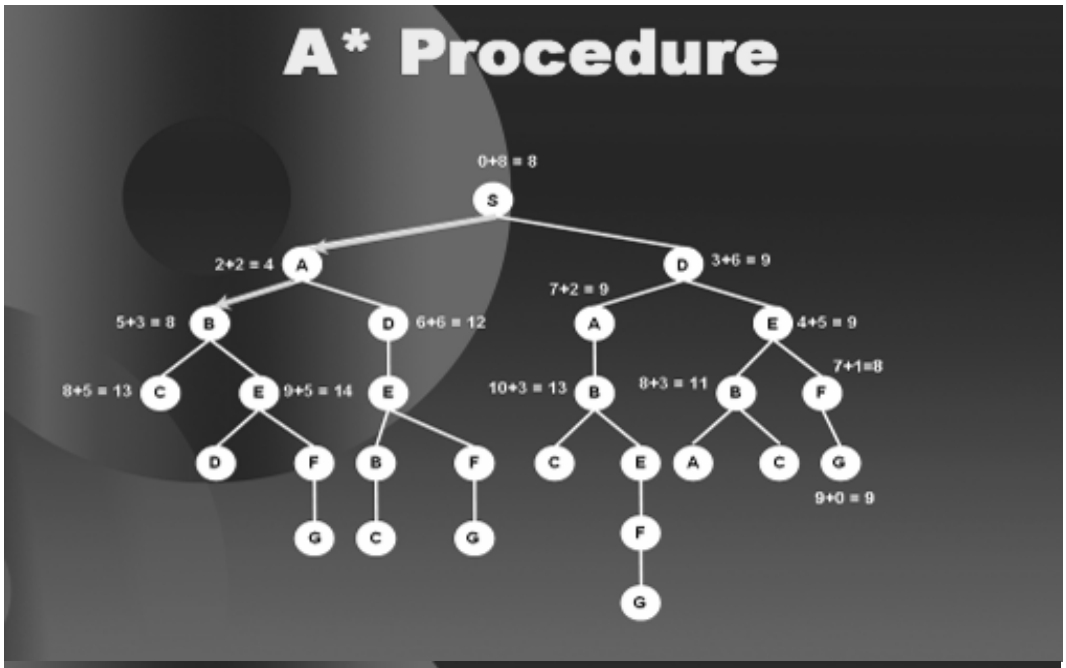
Our measure of goodness and badness of a node will now be decided by a combination of values that is the distance traveled so far and the estimate of the remaining distance. We construct the tree corresponding to the graph above.
We start with a tree with goodness of every node mentioned on it.

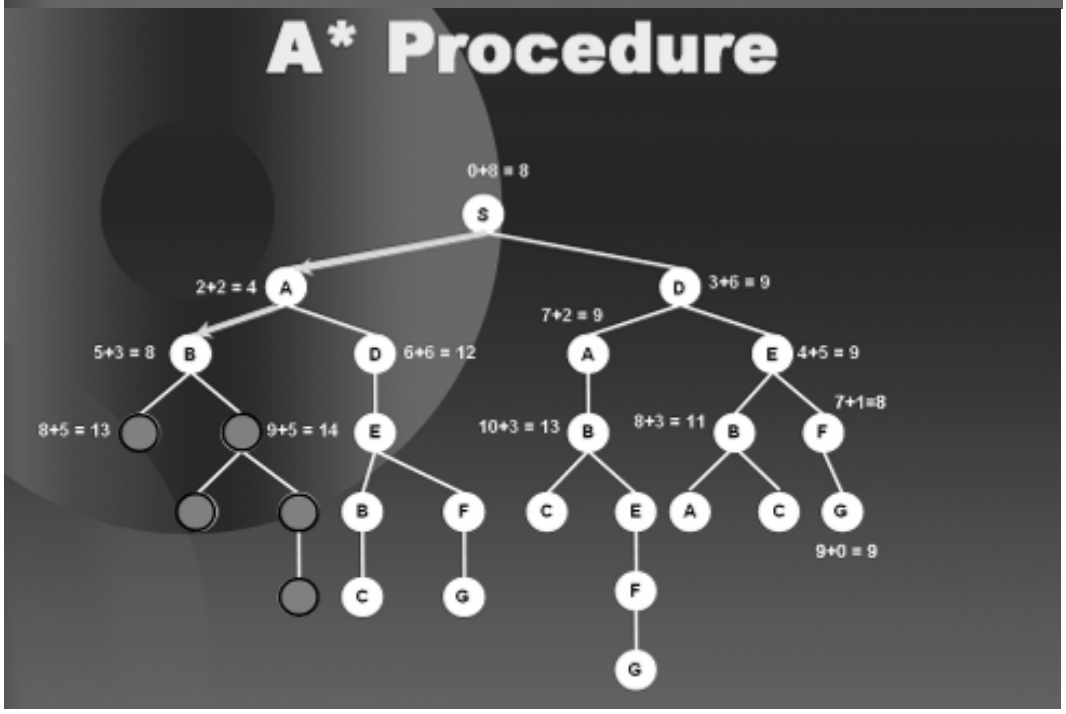


Standing at S we observe that the best node is A with a value of 4 so we move to 4.

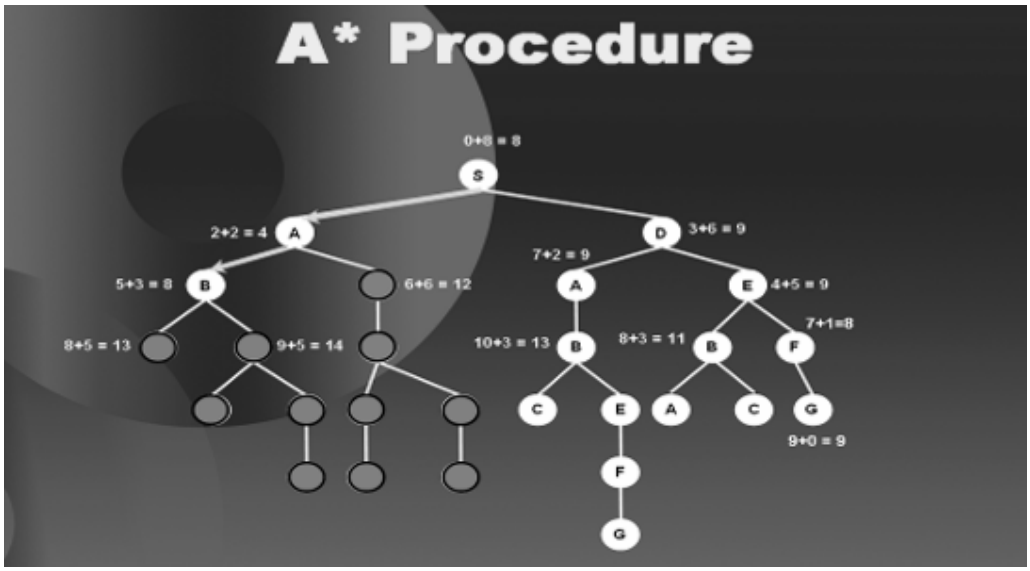


Then B. As all the sub-trees emerging from B make our path length more than 9 units so we bound this path, as shown in the next diagram.

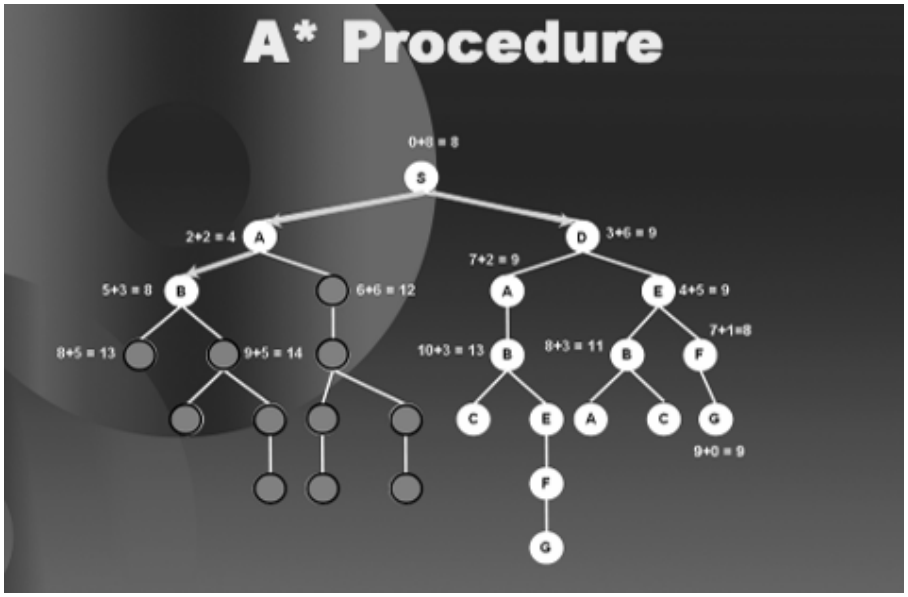




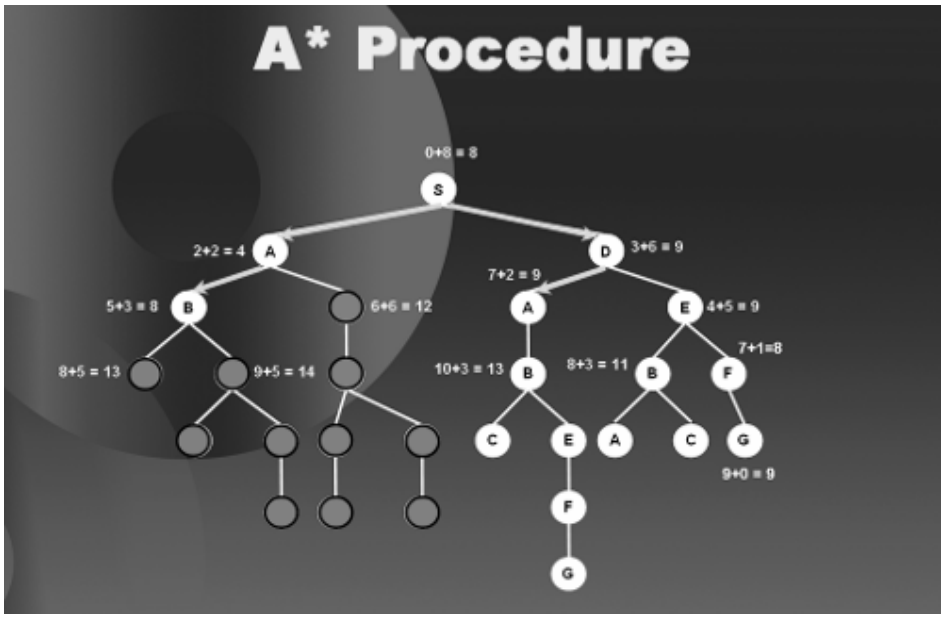
Now observe that to reach node D that is the child of A we can reach it either with a cost of 12 or we can directly reach D from S with a cost of 9. Hence using dynamic programming we will ignore the whole sub-tree beneath D (the child of A) as shown in the next diagram.



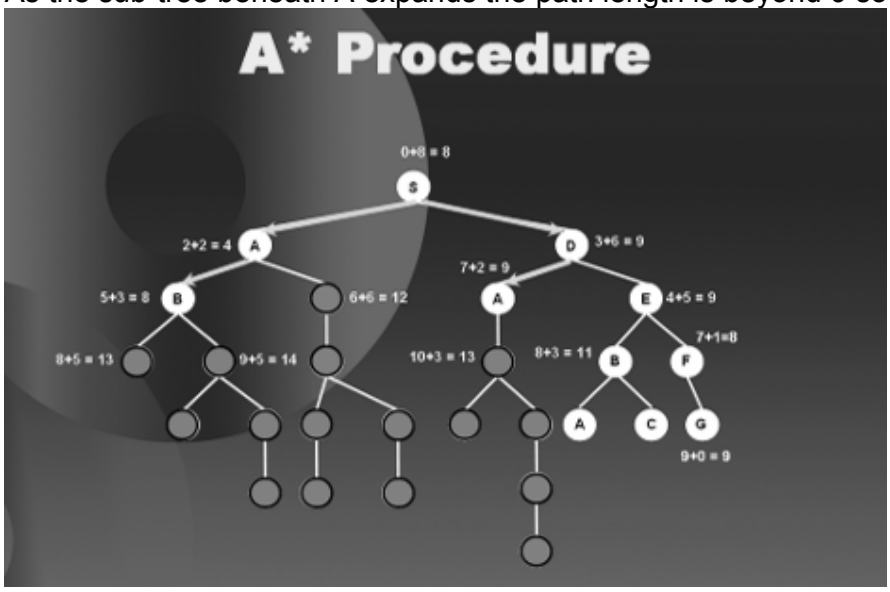
Now we move to D from S.



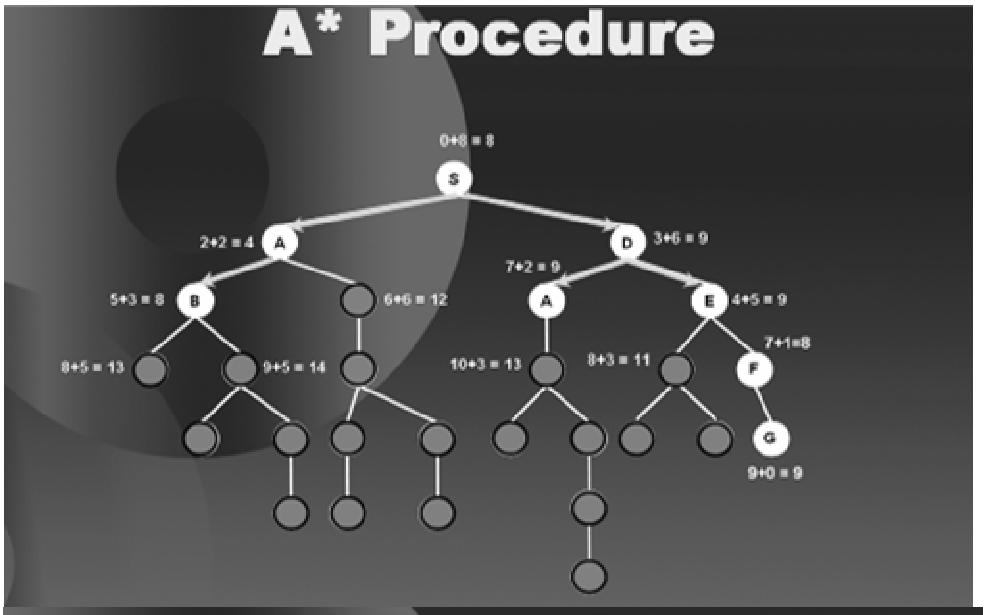
Now A and E are equally good nodes so we arbitrarily choose amongst them, and we move to A.

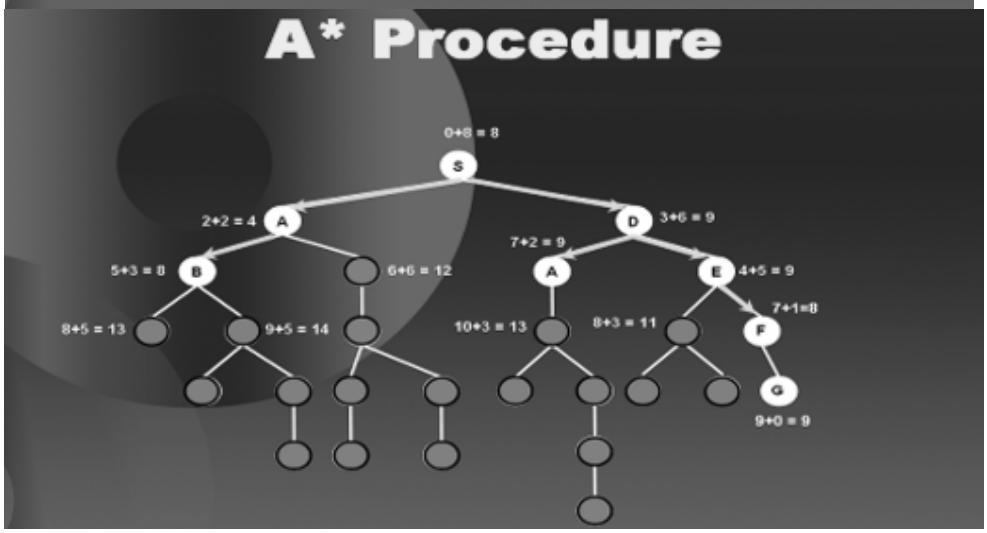


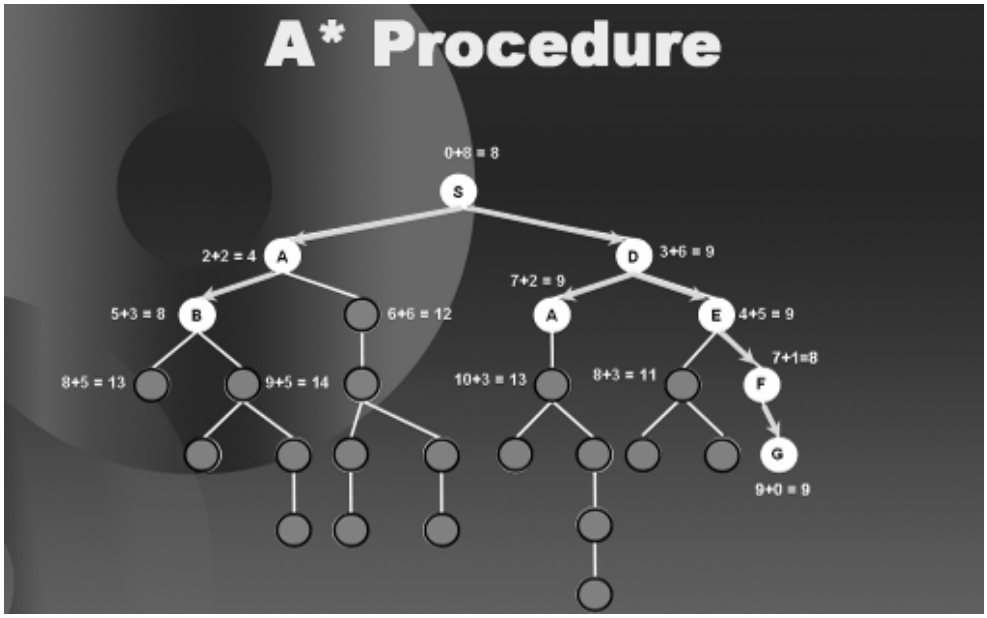
As the sub-tree beneath A expands the path length is beyond 9 so we bind it.



We proceed in this manner. Next we visit E, then we visit B the child of E, we bound the sub-tree below B. We visit F and finally we reach G as shown in the subsequent diagrams.





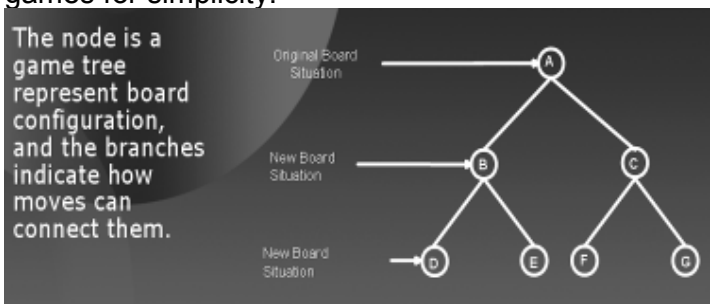


Notice that by using underestimates and dynamic programming the search space was further reduced and our optimal solution was found efficiently.

2.22 Adversarial Search

Up until now all the searches that we have studied there was only one person or agent searching the solution space to find the goal or the solution. In many applications there might be multiple agents or persons searching for solutions in the same solution space.

Such scenarios usually occur in game playing where two opponents also called adversaries are searching for a goal. Their goals are usually contrary to each other. For example, in a game of tic-tac-toe player one might want that he should complete a line with crosses while at the same time player two wants to complete a line of zeros. Hence both have different goals. Notice further that if player one puts a cross in any box, player-two will intelligently try to make a move that would leave player-one with minimum chance to win, that is, he will try to stop player-one from completing a line of crosses and at the same time will try to complete his line of zeros.
Many games can be modeled as trees as shown below. We will focus on board games for simplicity.



Searches in which two or more players with contrary goals are trying to explore the same solution space in search of the solution are called adversarial searches.

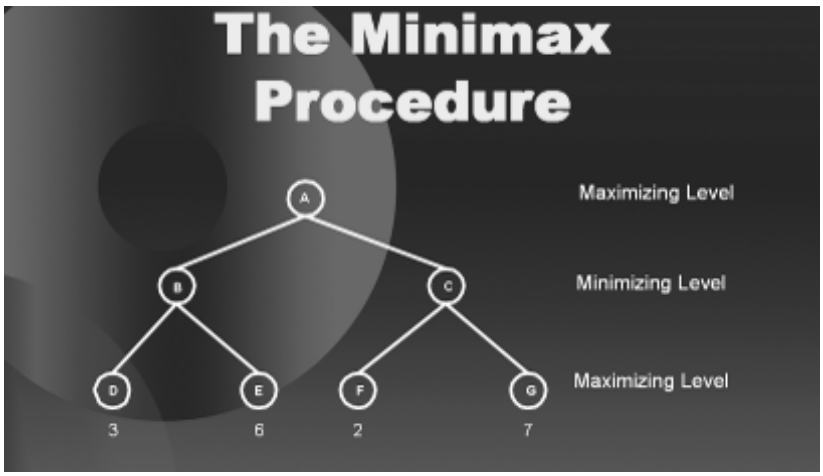
2.23 Minimax Procedure

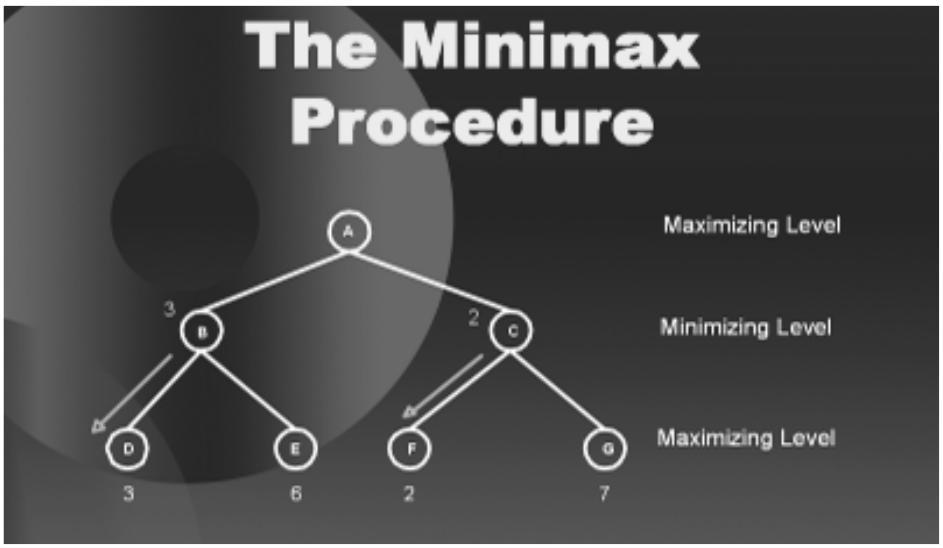
In adversarial searches one player tries to cater for the opponent's moves by intelligently deciding that what will be the impact of his own move on the over all configuration of the game. To develop this stance he uses a look ahead thinking strategy. That is, before making a move he looks a few levels down the game tree to see that what can be the impact of his move and what options will be open to the opponent once he has made this move.

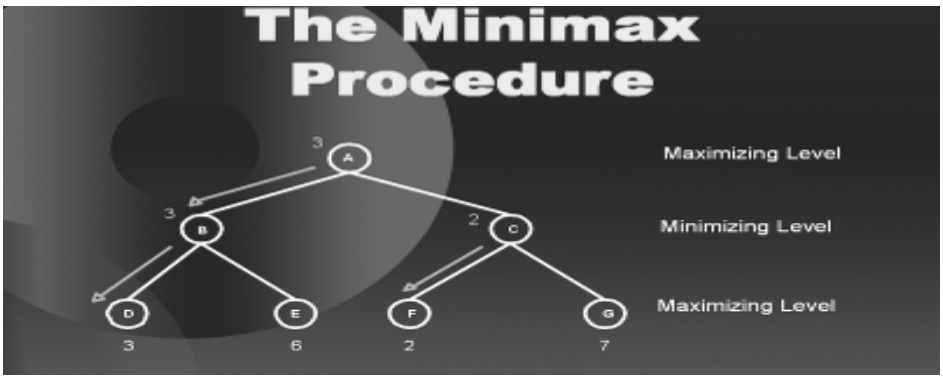
To clarify the concept of adversarial search let us discuss a procedure called the minimax procedure.

Here we assume that we have a situation analyzer that converts all judgments about board situations into a single, over all quality number. This situation analyzer is also called a static evaluator and the score/ number calculated by the evaluator is called the static evaluation of that node. Positive numbers, by convention indicate favor to one player. Negative numbers indicate favor to the other player. The player hoping for positive numbers is called maximizing player or maximizer. The other player is called minimizing player or minimizer. The maximizer has to keep in view that what choices will be available to the minimizer on the next step. The minimizer has to keep in view that what choices will be available to the maximizer on the next step.

Consider the following diagram.







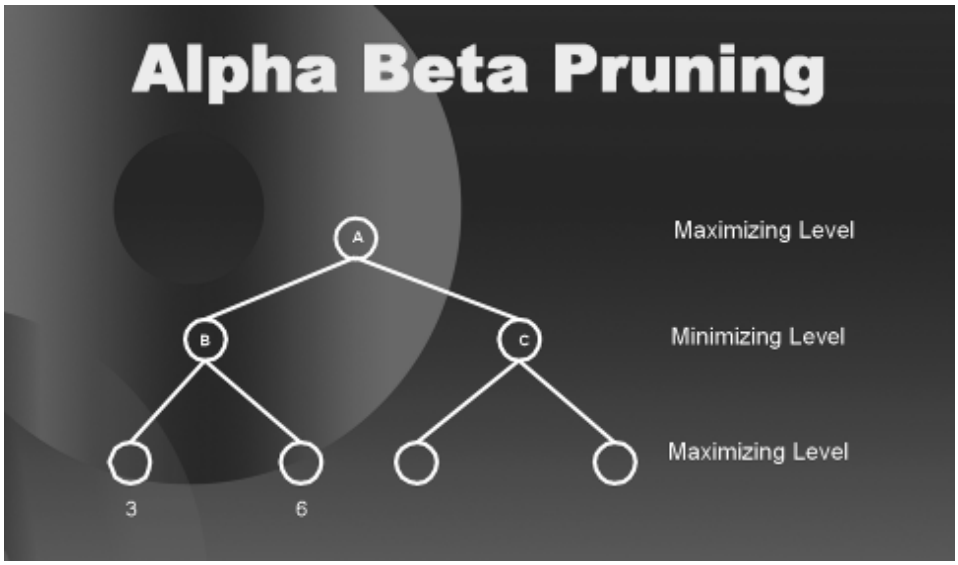
Standing at node A the maximizer wants to decide which node to visit next, that is, choose between B or C. The maximizer wishes to maximize the score so apparently 7 being the maximum score, the maximizer should go to C and then to G. But when the maximizer will reach C the next turn to select the node will be of the minimizer, which will force the game to reach configuration/node F with a score of 2. Hence maximizer will end up with a score of 2 if he goes to C from A. On the other hand, if the maximizer goes to B from A the worst which the minimizer can do is that he will force the maximizer to a score of 3. Now, since the choice is between scores of 3 or 2, the maximizer will go to node B from A.

2.24 Alpha Beta Pruning

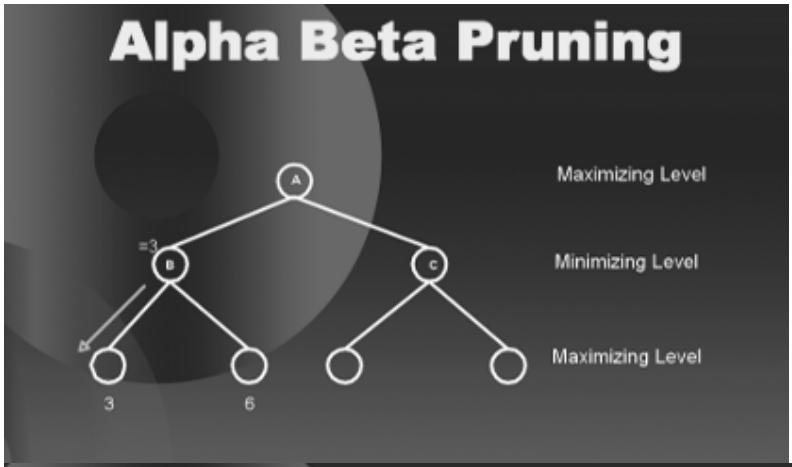
In Minimax Procedure, it seems as if the static evaluator must be used on each leaf node. Fortunately there is a procedure that reduces both the tree branches that must be generated and the number of evaluations. This procedure is called Alpha Beta pruning which “prunes” the tree branches thus reducing the number of static evaluations.

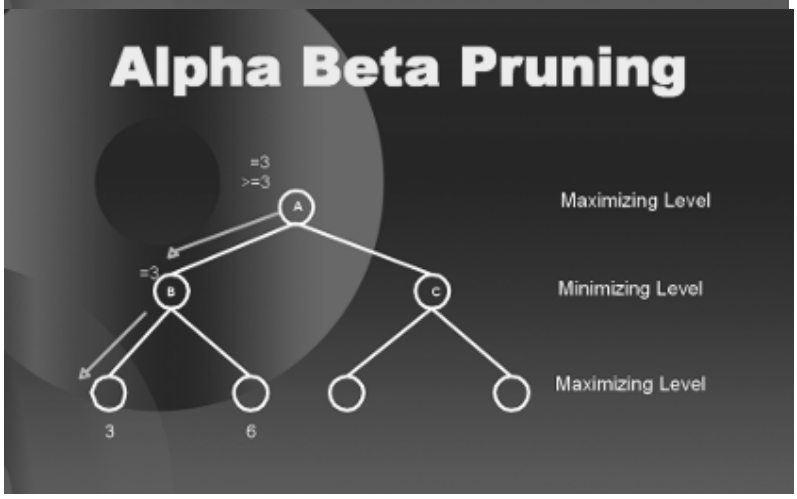
We use the following example to explain the notion of Alpha Beta Pruning.

Suppose we start of with a game tree in the diagram below. Notice that all nodes/situations have not yet been previously evaluated for their static evaluation score. Only two leaf nodes have been evaluated so far.

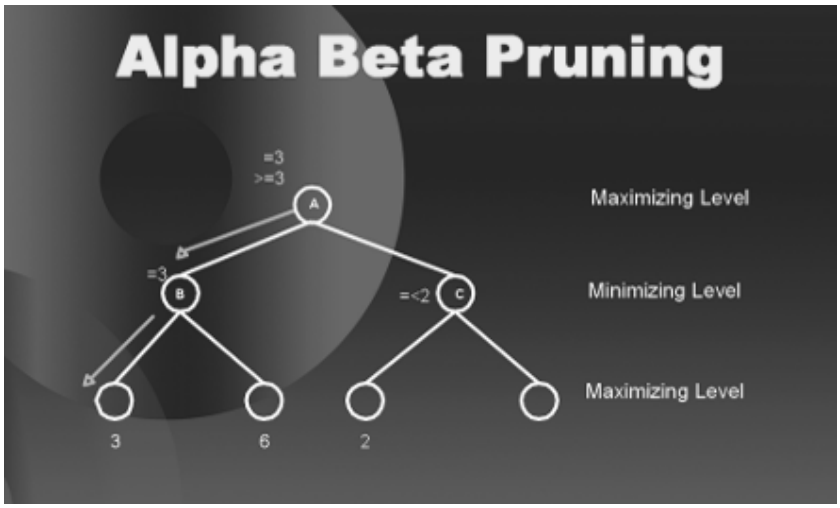


Sitting at A, the player-one will observe that if he moves to B the best he can get is 3.



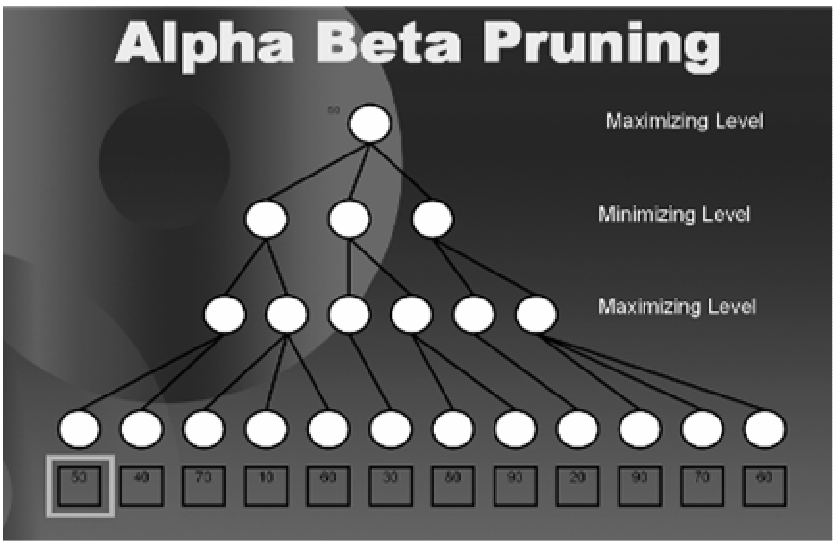


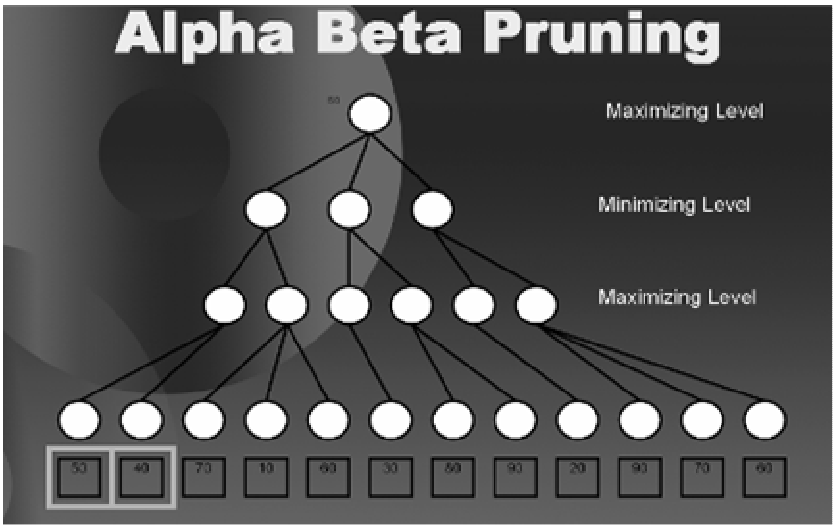
So the value three travels to the root A. Now after observing the other side of the tree, this score will either increase or will remain the same as this level is for the maximizer.

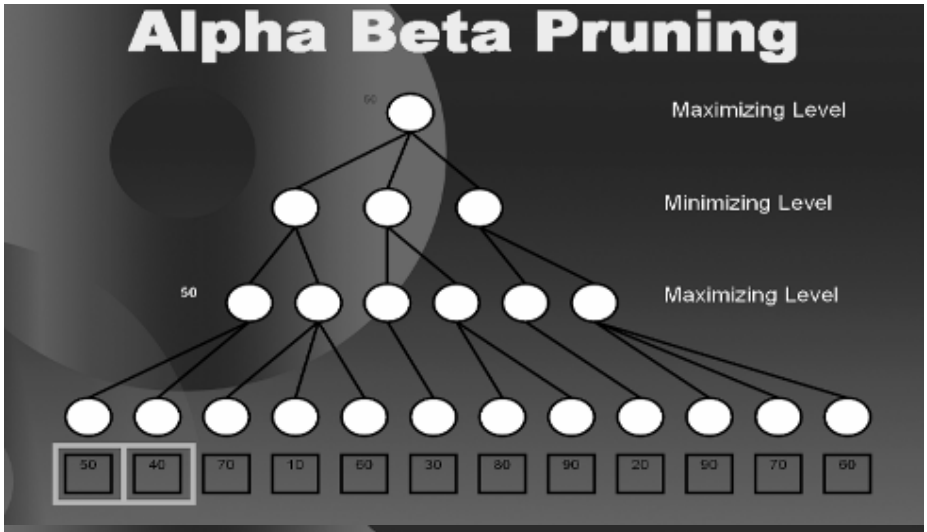


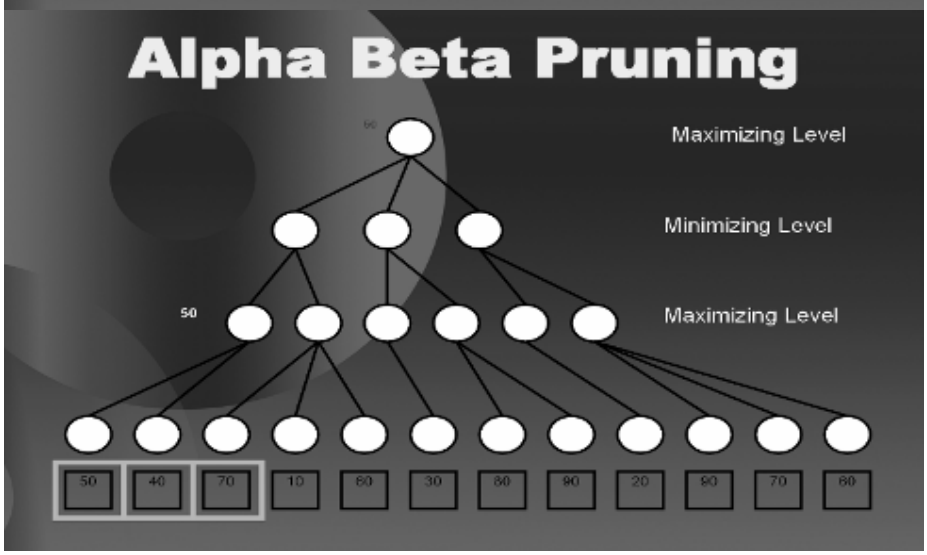
When he evaluates the first leaf node on the other side of the tree, he will see that the minimizer can force him to a score of less than 3 hence there is no need to fully explore the tree from that side. Hence the right most branch of the tree will be pruned and won't be evaluated for static evaluation.

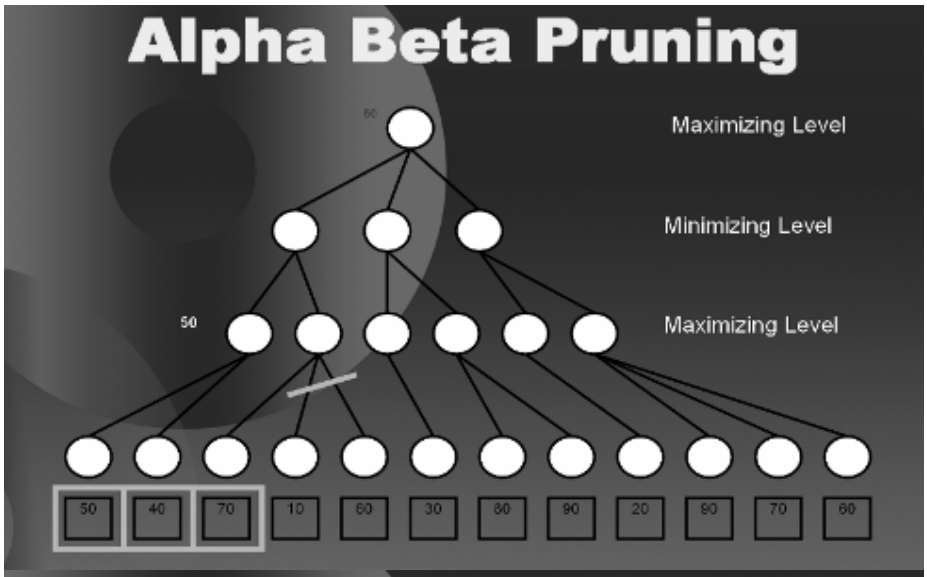
We have discussed a detailed example on Alpha Beta Pruning in the lectures. We have shown the sequence of steps in the diagrams below. The readers are required to go though the last portion of Lecture 10 for the explanation of this example, if required.

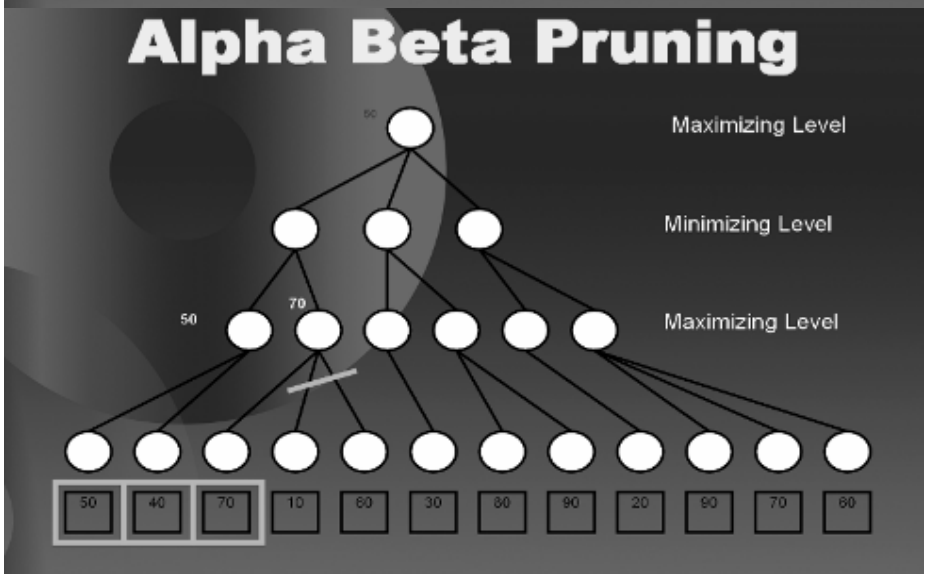


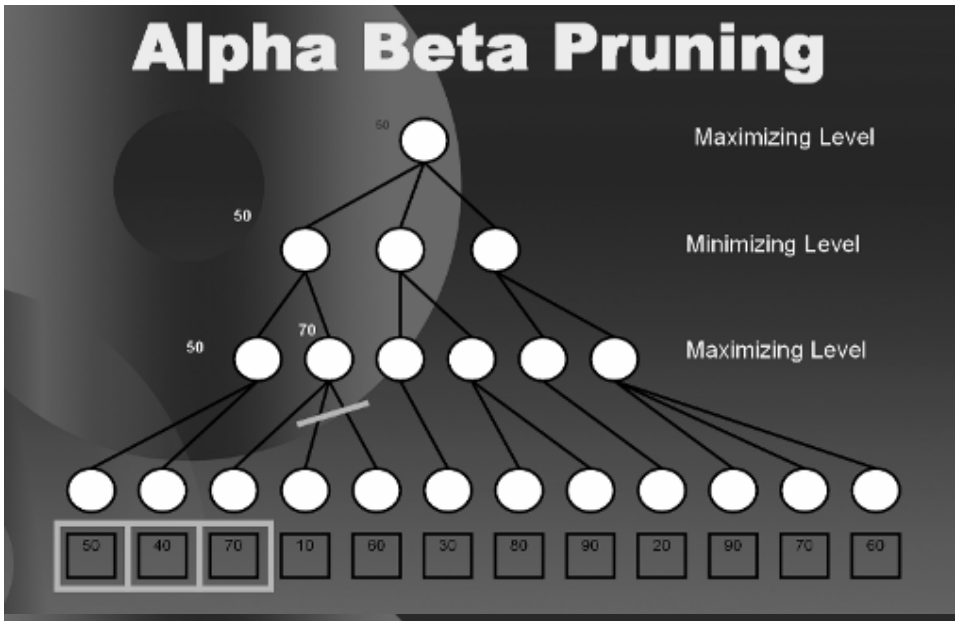


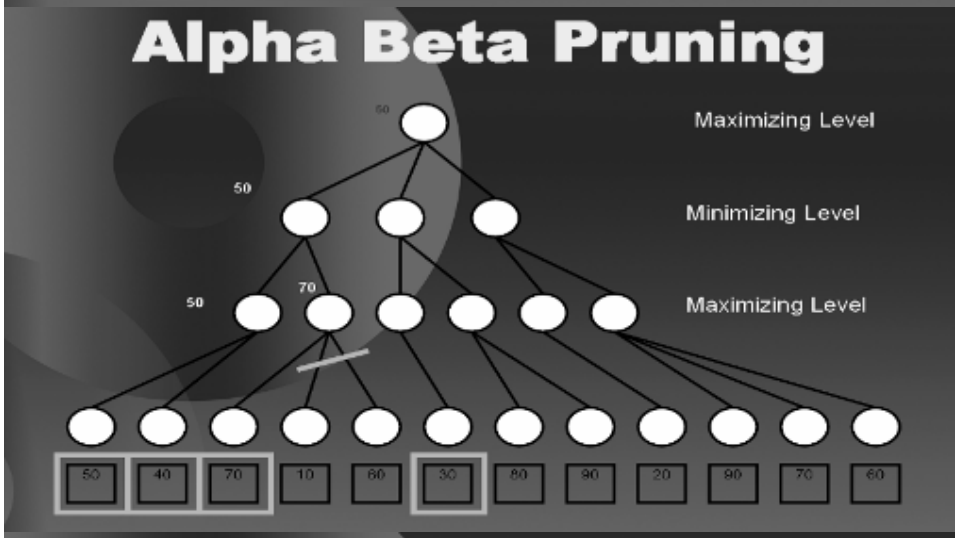


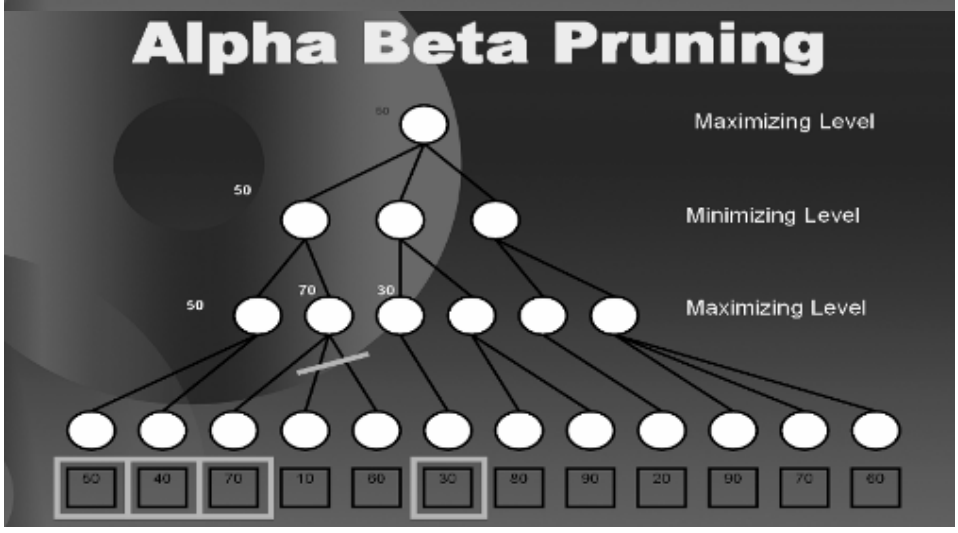


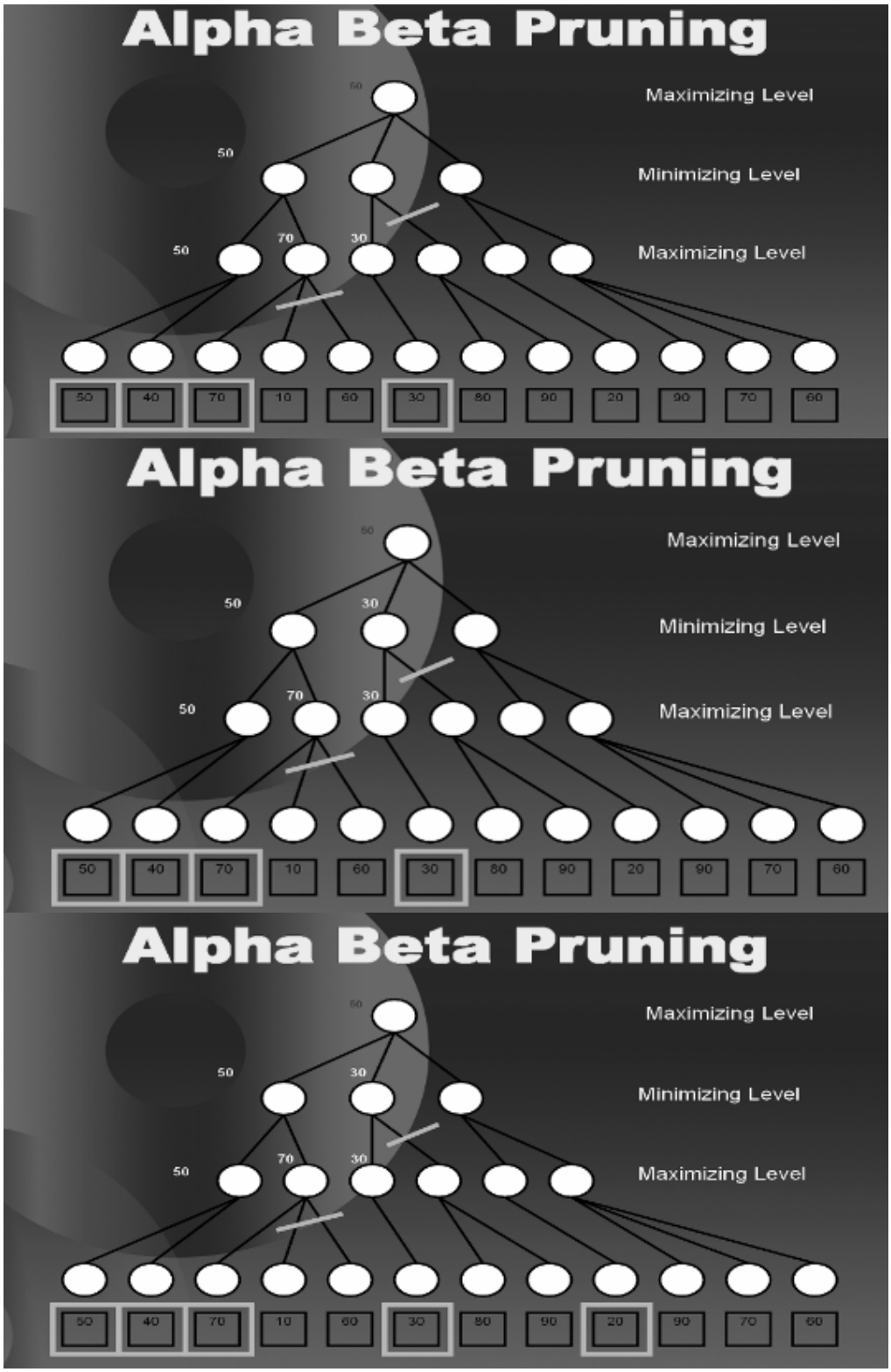


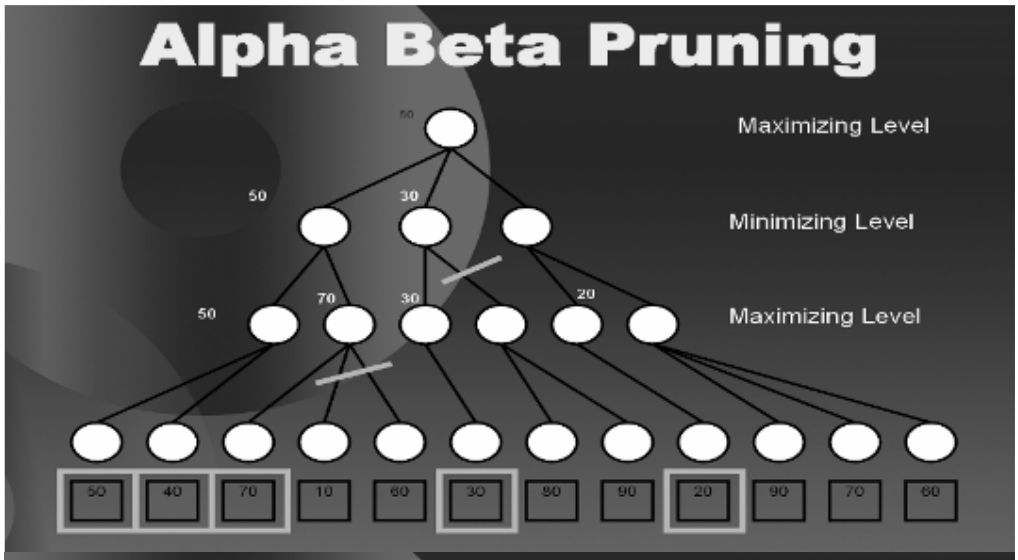


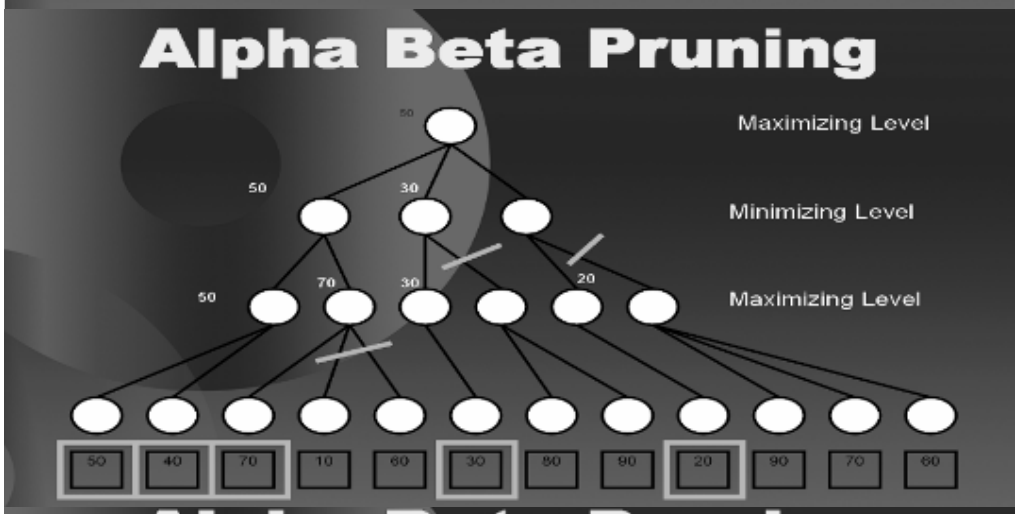


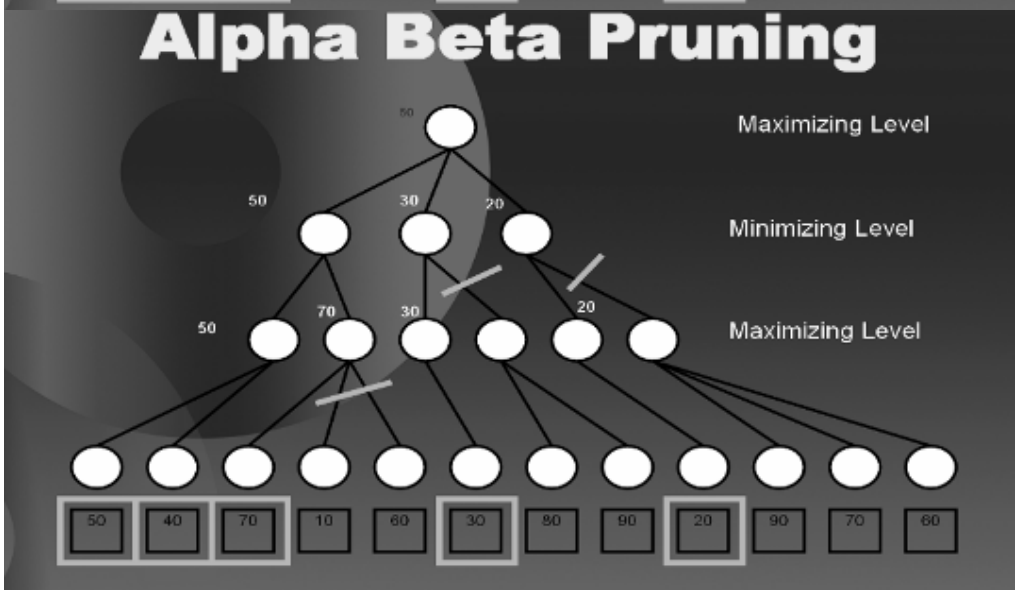












2.25 Summary

- People used to think that one who can solve more problems is more intelligent

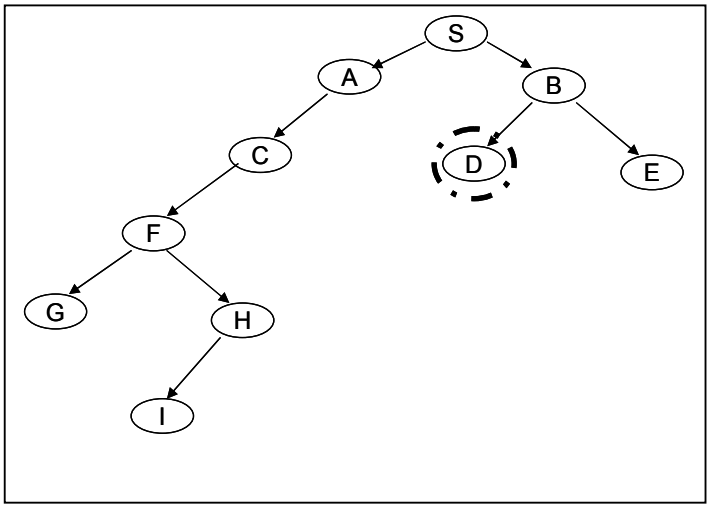
- Generate and test is the classical approach to solving problems
- Problem representation plays a key role in problem solving
- The components of problem solving include
  - Problem Statement
  - Operators
  - Goal State
  - Solution Space
- Searching is a formal mechanism to explore alternatives
- Searches can be blind or uninformed, informed, heuristic, non-optimal and optional.
- Different procedures to implement different search strategies form the major content of this chapter

2.26 Problems

Q1 Consider that a person has never been to the city air port. Its early in the morning and assume that no other person is awake in the town who can guide him on the way. He has to drive on his car but doesn't know the way to air port. Clearly identify the four components of problem solving in the above statement, i.e. problem statement, operators, solution space, and goal state. Should he follow blind or heuristic search strategy? Try to model the problem in a graphical representation.

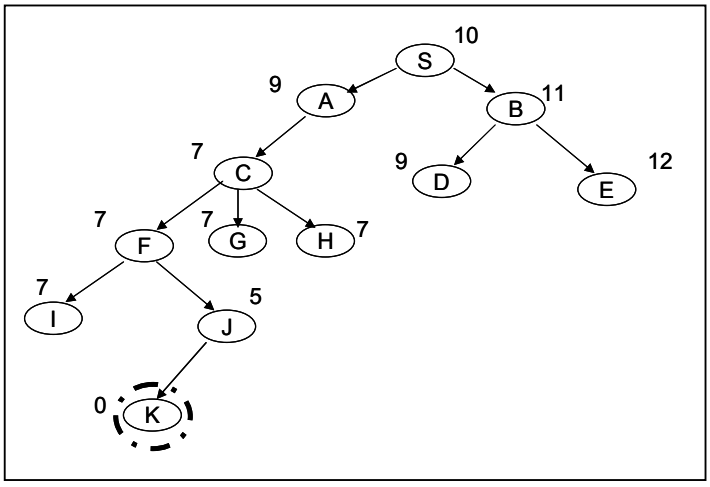
Q2 Clearly identify the difference between WSP (Well-Structured Problems) and ISP (Ill- Structured) problems as discussed in the lecture. Give relevant examples.

Q3 Given the following tree. Apply DFS and BFS as studied in the chapter. Show the state of the data structure **Q** and the visited list clearly at every step. S is the initial state and D is the goal state.



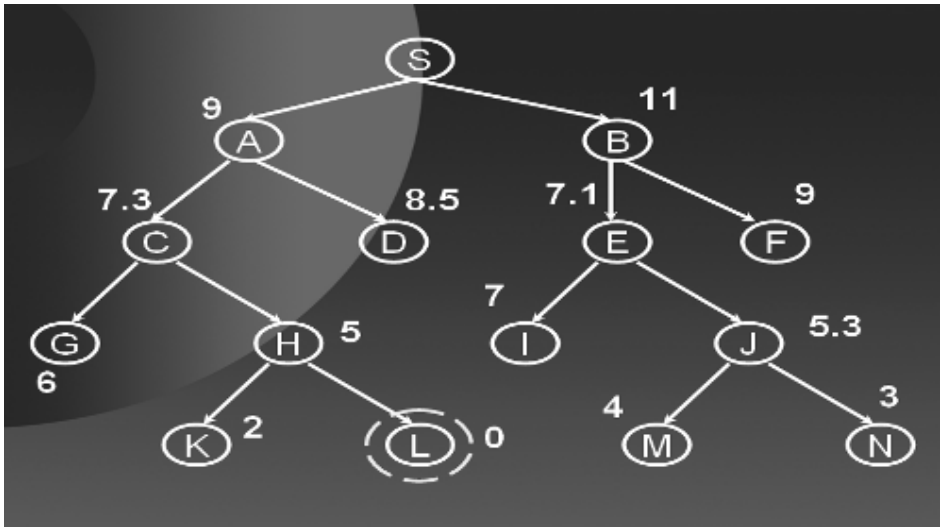
Q4 Discuss how progressive deepening uses a mixture of DFS and BFS to eliminate the disadvantages of both and at the same time finds the solution is a given tree. Support your answer with examples of a few trees.

Q5 Discuss the problems in Hill Climbing. Suggest solutions to the commonly encountered problems that are local maxima, plateau problem and ridge problem. Given the following tree, use the hill climbing procedure to climb up the tree. Use your suggested solutions to the above mention problems if any of them are encountered. K is the goal state and numbers written on each node is the estimate of remaining distance to the goal.

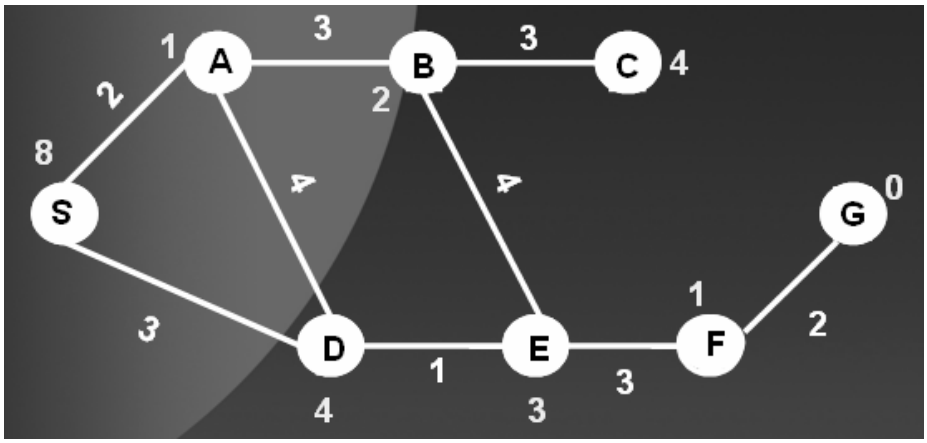


Q6 Discuss how best first search works in a tree. Support your answer with an example tree. Is best first search always the best strategy? Will it always guarantee the best solution?

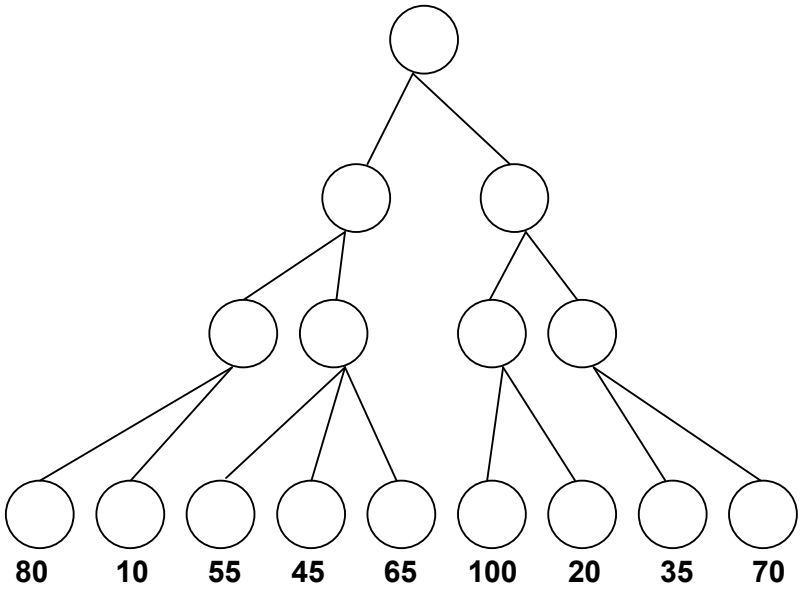
Q7 Discuss how beam search with degree of the search = 3 propagates in the given search tree. Is it equal to best first search when the degree = 1.



Q8 Discuss the main concept behind branch and bound search strategy. Suggest Improvements in the Algorithm. Simulate the algorithm on the given graph below. The values on the links are the distances between the cities. The numbers on the nodes are the estimated distance on the node from the goal state.

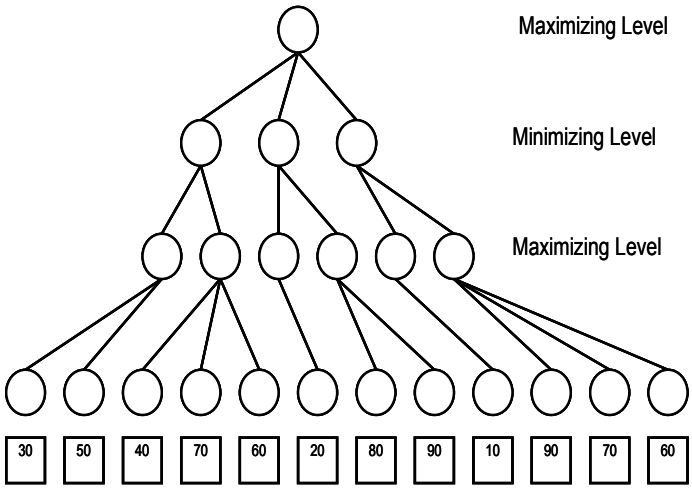


Q9. Run the MiniMax procedure on the given tree. The static evaluation scores for each leaf node are written under it. For example the static evaluation scores for the left most leaf node is 80.



Q10 Discuss how Alpha Beta Pruning minimizes the number of static evaluations at the leaf nodes by pruning branches. Support your answer with small examples of a few trees.

Q11 Simulate the Minimax procedure with Alpha Beta Pruning algorithm on the following search tree.



Adapted from: Artificial Intelligence, Third Edition by Patrick Henry Winston